1 Introduction

Nowadays, almost all of us carry mobile devices that communicate to each other in a direct or indirect way. Mobile ad-hoc network (MANET) is a decentralized network which comprises of a collection of these devices/nodes that interact with each other, forming a temporary network without any aid of a centralized administration. Since the advent of it, MANETs are now integral components of many applications such as military applications, virtual conferences, and disaster recovery operations; they have become part of out daily lives.

The topology of a MANET is dynamic as the nodes belonging to MANET can enter or leave constantly. Devices, we call them as *nodes* henceforth, must route the packets to another in order to make the network fully connected. Two chosen nodes are said to be 'directly connected' to each other as long as they are located somewhere within the transmission ranges of each other so they can communicate directly; otherwise they would require other nodes to communicate. Hence, a route should be established first for the message delivery between two nodes.

Routing –a process of moving a packet of data from source to destination– in such dynamic, infrastructure-less mobile ad-hoc network has always been an attractive research area for scientists, it is known as **routing in MANET** in the literature. The quality of a route between sender and receiver nodes varies depending on the network or the problem type at hand.

In the following section, approaches for network and route construction as well as optimal route selection are described in detail.

2 AdHocSim Simulator

In this assignment, you are expected to implement a program that simulates MANET, we call simulator as **AdHocSim**. You first need to establish the network, then find possible routes between the sender and receiver nodes, and finally choose the optimal route among them. To accomplish this task you have to use **recursion** while constructing the possible routes.

In Section 2.1, how to construct nodes and determine their neighbors are explained; Section 2.2 describes the approach that you need to consider for finding possible routes. Finally, the selection criteria for choosing optimal route among possible routes are detailed in Section 2.3.

2.1 Setting-up the Nodes and the Neighbors

As mentioned, MANET comprises of nodes that communicate each other directly or indirectly by employing other nodes as router nodes in the case of they are not located within the communication range. In this assignment, a node carries three different information: i) *location*, ii) *transmission range*, and iii) *residual battery level*. For example, node A is located at `(x=120,y=45)` and has 80% battery level (see Fig. 1).

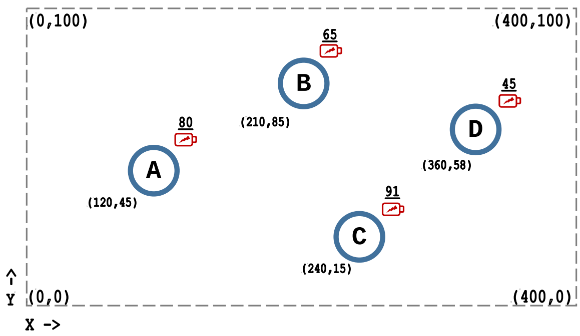


Figure 1: Nodes in MANET

To determine the neighborhood of each node, you first need to examine the **location** and **transmission range** information. To say A is one of the neighbors of B, A should be within the transmission range of B. As mentioned, every node has location and transmission range where coordinate limits are characterized by *east* (x_1), *west* (x_2), *north* (y_1), and *south* (y_2). For example `B` and `C` are the two neighbors of `A` (see Fig. 2).

Note: To represent the neighborhood, you are advised to construct a **dictionary** where *key* is the node label and its corresponding *value* is its neighbors, as illustrated below:

```
'A': ['B', 'C']
'B': ['D', 'E']
'C': ['F', 'H']
'D': ['E', 'G']
'E': ['G']
'F': ['G', 'H']
'G': ['I']
'H': []
'I': []
```

2.2 Route Finding in MANET

You may think of a network as a directed graph where nodes represent vertices whereas communication links represent edges. A graph might be *cyclic* or *acyclic*. A graph is said to be cyclic if there exist some vertices that are connected in a closed chain (see Fig. 2 for an example of an acyclic graph). In this assignment, you will obtain cyclic or acyclic network once you set-up nodes and their neighbors successfully. Concretely speaking, there might be a case where the next node is a node that has already been **visited before**. In such a case, you should design your program such that it **refuses to visit such nodes again and again** in order to avoid the infinite loop problem.

After setting up the nodes and determining of neighbors of every node, we obtain a network ready-to-use. Before transmission of first packet begins, it is needed to **find a path** through which packets are transferred. This path is called as *route* that starts and ends with a sender and destination nodes, respectively. Remember that route consists of only sender and

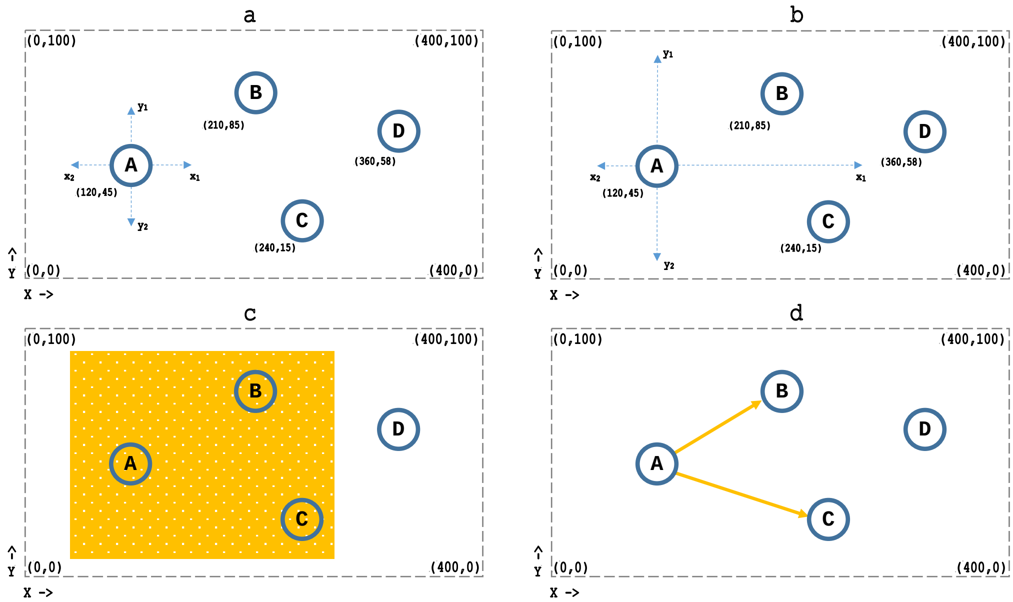


Figure 2: Determining relations in a MANET

destination nodes if they are within each other's transmission range. But, on the other, they employ other nodes as intermediate nodes by treating them routers for transmission of a collection of packets.

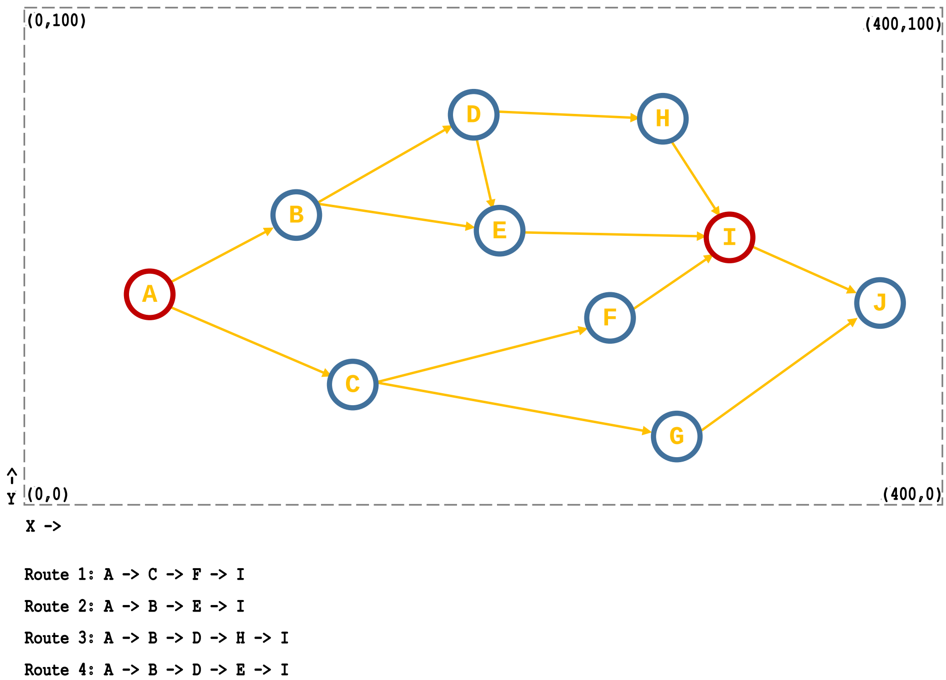


Figure 3: Steps of route finding process in a MANET

To clarify the routing process in MANET, we make use of Fig. 3. From the figure, sender node `A` is ready to initiate a data transfer to destination node `I` and there exist four routes between `A` and `I`. (Note: There might be cases where there is no route between any given two nodes.) A step by step route finding process is demonstrated with figures at the end of this document. With a similar way, you are expected to find every possible routes, using **recursion**, with its cost whose calculation is explained in the following section.

2.3 Selecting an Optimal Route

This process is similar to what actually navigation application does. Suppose Alice wants to visit Bob and uses his application to take a route to Bob's house. Application returns Alice's request with a route within a series of possible routes, which is often shortest path with respect to the *time* or *distance*. In this scenario, time and distance are the two costs that determine the quality of the routes. Similar to this scenario, one needs to determine one of the routes to use for packet transmission depending on the their costs after finding all routes. The cost of a route is the sum of the cost of every communication links (L) between nodes (see Eq. 1 and Fig. 4).

$$Cost(Route_r) = \sum Cost(i,j) \quad | \quad i,j \in L \tag{1}$$

In this assignment, you will consider *distance* and *residual battery level* for the calculation of cost of a link between nodes i and j given by:.

$$Cost(i,j) = \frac{dist_{i,j}}{battery_j} \tag{2}$$

where $dist_{i,j}$ is a distance between the nodes i and j. As these nodes are actually represented by two points on the Cartesian plane (i_x and i_y are two points of node i whereas j_x and j_y are the two points of node j), you can get the distance between them using Euclidean distance:

$$dist_{i,j} = \sqrt{(i_x - j_x)^2 + (i_y - j_y)^2} \tag{3}$$

In MANETs, apart from initial data transfer, it is also necessary to find routes every time a link breakage occurs. Link breakage problem arises mostly when a node leaves the route through which data packets are transferred. A node becomes unreachable either if it runs out of its battery or goes beyond the transmission range of a node from which it receives packets. That's why distance and residual battery level has been intentionally included in the calculation of every link cost in this assignment. The selection of the optimal route ($Route_{OPT}$) among the possible routes (R) is made as follows (by adopting this, you ensure that you select a link that has a closer link and a higher residual battery):

$$Route_{OPT} = \arg\min_{r} \; Cost(Route_r) \mid r \in R \tag{4}$$

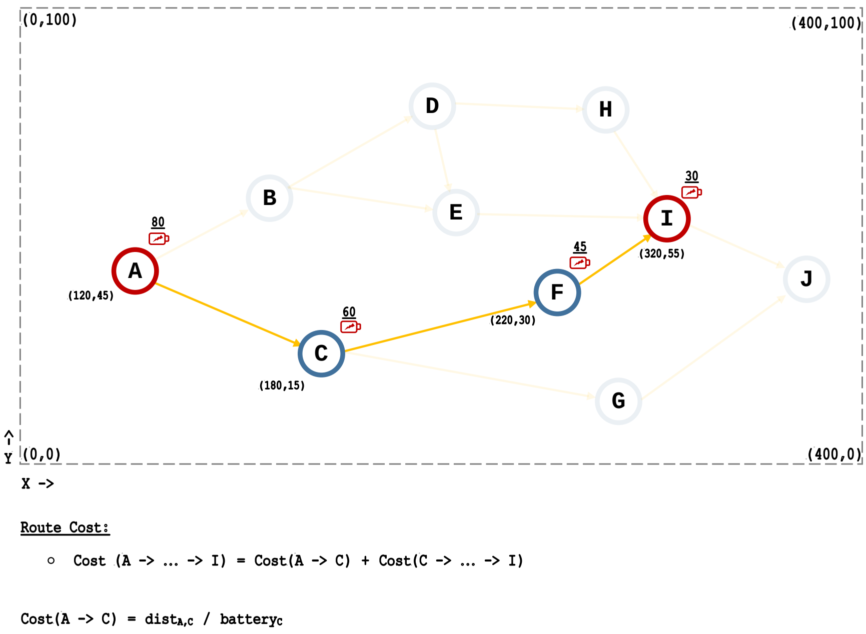


Figure 4: An example of a route cost

2.4 Selective Forwarding Attack

There is a variety of routing attacks that have been developed to compromise the packet delivery performance of the network. Blackhole attack, sinkhole attack, sybil attack, selective forwarding attack, and etc. are the well-known attack types targeting the data packets to manipulate as they wish.

In this assignment, you are expected to concentrate on only selective forwarding attack. The intruder/malicious nodes in selective forwarding attack behave like normal nodes but selectively drop the packets, which in turn increases the overall drop rate and end-to-end delay (E2E) of the network. In this assignment, assume that there may (1 node at most) or may not be an intruder/malicious node. The intruder node **drops one packet after every three packets**, namely drop rate will be 25%.

To handle this type of routing attack in this assignment, you need to reestablish routing from source to destination, but **excluding the intruder node**. This should be in a way that it *i*) counts the drop times of every node and *ii*) suspends the data transfer once the counter reaches **two drops**, then finds other routes to the same destination, and finally sends the remaining packets.

2.5 Simulator

The simulator you are asked to implement should start the simulation just after the network is established in line with the commands provided in a file named **commands.txt** and should terminate itself once the whole data transfer is completed. Here, you need to evaluate six types of command:

- CRNODE (Creates a new node): It generates a new node and requires four arguments to work properly. The first argument gives node a label (say i); second argument defines its locations as x and y coordinates (i.e., i_x and i_y); transmission range is determined in third argument in order of (x_1;x_2;y_1;y_2). Finally, residual battery level is set by forth argument.

  ```
  CRNODE y 120;40 100;0;60;40 65
  ```

  Every time you evaluate CRNODE command, the following text should be displayed on the screen:

  ```
  COMMAND *CRNODE*: New node y is created
  ```

- SEND (Sends data): It initiates data transfer from a source node to a destination node. It takes three arguments; the first two defines source and destination node labels, respectively. The last argument is the amount of data (in Byte) to be transferred.

  ```
  SEND x y 1002
  ```

  Every time you evaluate SEND command the following text should be displayed on the screen:

  ```
  COMMAND *SEND*: Data is ready to send from x to y
  ```

- MOVE (Moves node): It relocates a node to a pair of points that is defined as an argument provided with itself. There exist two arguments first of which is the node label whereas other defines its new location as x and y coordinates.

  ```
  MOVE x 165;70
  ```

  Every time you evaluate MOVE command the following text should be displayed on the screen:

  ```
  COMMAND *MOVE*: The location of node x is changed
  ```

- CHBTTRY (Changes battery level): It simply charges or discharges the battery of a given node. It takes two arguments: i) node label and ii) new battery level.

  ```
  CHBTTRY x 90
  ```

  Every time you evaluate CHBTTRY command the following text should be displayed on the screen:

  ```
  COMMAND *CHBTTRY*: Battery level of node x is changed to 90
  ```

- RMNODE (Removes an existing node): It vanishes an existing node that is determined by an argument.

  ```
  RMNODE y
  ```

  Every time you evaluate RMNODE command the following text should be displayed on the screen:

  ```
  COMMAND *RMNODE*: Node y is removed
  ```

- INTRUDE (Launches selective forwarding attack): It transforms a node into the intruder node.

```
INTRUDE y
```

At a time when you evaluate INTRUDE command the following text should be displayed on the screen:

```
COMMAND *INTRUDE*: Node y has become a malicious node
```

Note: The commands and their parameters are separated by `<TAB>` characters, and each parameter is also separated from each other by `<TAB>` characters.

You will notice one more argument at the beginning of every line in the **commands.txt** file. It represents the time (in Second) that points out when a command of interest should be applied. For example, '`20 CHBTTRY x4 90`' command line should be evaluated such that the residual battery level of node `x4` will be set to `90` when the simulator reaches 20th second. The input file (i.e., **commands.txt**) will always begin with SEND and CRNODE commands so that you configure the network (see Section 2.1) and begin transmission of data through packets.

Note: Any time a node is **created**, **removed**, **re-localized**, the **battery level of it is changed**, or **the attack is detected**; you need to initiate the route-finding process from scratch, which is naturally expected. Once at least one of these cases occurs, you must print out *i*) relations in the network, *ii*) possible routes found, and *iii*) the optimal route chosen in the following format:

```
NODES & THEIR NEIGHBORS: x1 -> x2, x3 | x2 -> x4, x5 | x3 -> x4 | x4 -> x5, x7 | x5 -> x7 | x6
-> x7, x8 | x7 -> x9 | x8 ->  | x9 ->  |
5 ROUTE(S) FOUND:
ROUTE 1: x1 -> x2 -> x4 -> x5 -> x7 -> x9  COST: 4.9219
ROUTE 2: x1 -> x2 -> x4 -> x7 -> x9  COST: 5.0150
ROUTE 3: x1 -> x2 -> x5 -> x7 -> x9  COST: 4.1031
ROUTE 4: x1 -> x3 -> x4 -> x5 -> x7 -> x9  COST: 4.5047
ROUTE 5: x1 -> x3 -> x4 -> x7 -> x9  COST: 4.5978
SELECTED ROUTE (ROUTE 3): x1 -> x2 -> x5 -> x7 -> x9
```

Note: A path might not be found between endpoints at every time. In such case simulator should terminate itself just after the message below is displayed on the screen.

```
NO ROUTE FROM x TO y FOUND.
```

Before simulator starts, you need to calculate the total number of packets (NP) that are necessary to transfer whole data considering the equation below:

$$NP = \left\lceil \frac{DS}{PS} \right\rceil \tag{5}$$

where DS and PS are the data size and packet size, respectively. As mentioned DS is provided with SEND command (the last argument) while PS is provided as a **command-line argument**.

The last thing worth mentioning is that, in the AdHocSim simulator, **every simulation time corresponds to one second**, and **only one packet** is allowed to send from the source to the destination **in a single second**.

Note: You can design your implementation such that **every second** corresponds to **one individual step in the loop**. You need to print out time stamp at every simulation time in the following format:

```
SIMULATION TIME: <HH>:<MM>:<SS>
```

The command-line argument that runs your implementation should be as follows:

```
python AdHocSim.py PS
```

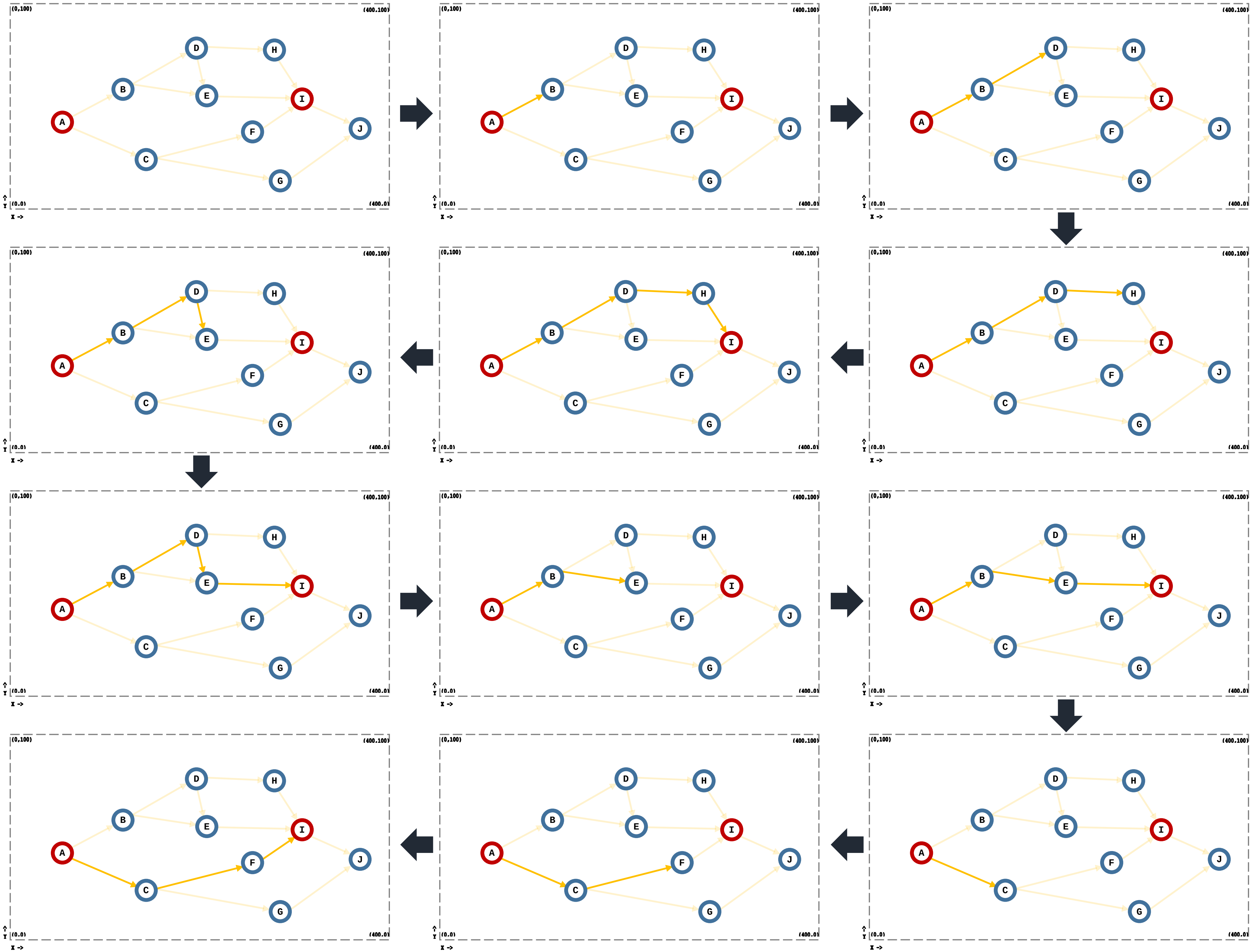
In the following page, an output sample that is created from the simulator when these commands are evaluated is given:

```
0 CRNODE x1 120;40 100;0;60;40 65
0 CRNODE x2 200;50 80;10;30;5 40
0 CRNODE x3 150;15 120;0;25;10 88
0 CRNODE x4 240;60 100;10;0;40 35
0 CRNODE x5 260;45 70;0;10;25 98
0 CRNODE x6 225;15 75;10;5;10 70
0 CRNODE x7 273;20 75;0;0;0 68
0 CRNODE x8 221;5 0;0;0;0 68
0 CRNODE x9 345;20 0;0;0;0 68
0 SEND x1 x9 477
10 MOVE x3 165;70
20 CHBTTRY x4 90
22 RMNODE x5
24 INTRUDE x3
```

Command-line argument:

```
python AdHocSim.py 15
```

```
*******************************
AD-HOC NETWORK SIMULATOR - BEGIN
*******************************
SIMULATION TIME: 00:00:00
	COMMAND *CRNODE*: New node x1 is created
	COMMAND *CRNODE*: New node x2 is created
	COMMAND *CRNODE*: New node x3 is created
	COMMAND *CRNODE*: New node x4 is created
	COMMAND *CRNODE*: New node x5 is created
	COMMAND *CRNODE*: New node x6 is created
	COMMAND *CRNODE*: New node x7 is created
	COMMAND *CRNODE*: New node x8 is created
	COMMAND *CRNODE*: New node x9 is created
	COMMAND *SEND*: Data is ready to send from x1 to x9
	PACKET 1 HAS BEEN SENT
	REMAINING DATA SIZE: 462.0 BYTE
	NODES & THEIR NEIGHBORS: x1 -> x2, x3 | x2 -> x4, x5 | x3 -> x6, x8 | x4 -> x5, x7 | x5 -> x7 | x6 -> x7, x8 | x7 -> x9 | x8 ->  | x9 ->  |
	4 ROUTE(S) FOUND:
	ROUTE 1: x1 -> x2 -> x4 -> x5 -> x7 -> x9	COST: 4.9219
	ROUTE 2: x1 -> x2 -> x4 -> x7 -> x9	COST: 5.0150
	ROUTE 3: x1 -> x2 -> x5 -> x7 -> x9	COST: 4.1031
	ROUTE 4: x1 -> x3 -> x6 -> x7 -> x9	COST: 3.2837
	SELECTED ROUTE (ROUTE 4): x1 -> x3 -> x6 -> x7 -> x9
SIMULATION TIME: 00:00:01
	PACKET 2 HAS BEEN SENT
	REMAINING DATA SIZE: 447.0 BYTE
SIMULATION TIME: 00:00:02
	PACKET 3 HAS BEEN SENT
	REMAINING DATA SIZE: 432.0 BYTE
SIMULATION TIME: 00:00:03
	PACKET 4 HAS BEEN SENT
	REMAINING DATA SIZE: 417.0 BYTE
SIMULATION TIME: 00:00:04
	PACKET 5 HAS BEEN SENT
	REMAINING DATA SIZE: 402.0 BYTE
SIMULATION TIME: 00:00:05
	PACKET 6 HAS BEEN SENT
	REMAINING DATA SIZE: 387.0 BYTE
SIMULATION TIME: 00:00:06
	PACKET 7 HAS BEEN SENT
	REMAINING DATA SIZE: 372.0 BYTE
SIMULATION TIME: 00:00:07
	PACKET 8 HAS BEEN SENT
	REMAINING DATA SIZE: 357.0 BYTE
SIMULATION TIME: 00:00:08
	PACKET 9 HAS BEEN SENT
	REMAINING DATA SIZE: 342.0 BYTE
SIMULATION TIME: 00:00:09
	PACKET 10 HAS BEEN SENT
	REMAINING DATA SIZE: 327.0 BYTE
SIMULATION TIME: 00:00:10
	COMMAND *MOVE*: The location of node x3 is changed
	PACKET 11 HAS BEEN SENT
	REMAINING DATA SIZE: 312.0 BYTE
	NODES & THEIR NEIGHBORS: x1 -> x2, x3 | x2 -> x4, x5 | x3 -> x4 | x4 -> x5, x7 | x5 -> x7 | x6 -> x7, x8 | x7 -> x9 | x8 ->  | x9 ->  |
	5 ROUTE(S) FOUND:
	ROUTE 1: x1 -> x2 -> x4 -> x5 -> x7 -> x9	COST: 4.9219
	ROUTE 2: x1 -> x2 -> x4 -> x7 -> x9	COST: 5.0150
	ROUTE 3: x1 -> x2 -> x5 -> x7 -> x9	COST: 4.1031
	ROUTE 4: x1 -> x3 -> x4 -> x5 -> x7 -> x9	COST: 4.5047
	ROUTE 5: x1 -> x3 -> x4 -> x7 -> x9	COST: 4.5978
	SELECTED ROUTE (ROUTE 3): x1 -> x2 -> x5 -> x7 -> x9
SIMULATION TIME: 00:00:11
	PACKET 12 HAS BEEN SENT
	REMAINING DATA SIZE: 297.0 BYTE
SIMULATION TIME: 00:00:12
	PACKET 13 HAS BEEN SENT
	REMAINING DATA SIZE: 282.0 BYTE
SIMULATION TIME: 00:00:13
	PACKET 14 HAS BEEN SENT
	REMAINING DATA SIZE: 267.0 BYTE
SIMULATION TIME: 00:00:14
	PACKET 15 HAS BEEN SENT
	REMAINING DATA SIZE: 252.0 BYTE
SIMULATION TIME: 00:00:15
	PACKET 16 HAS BEEN SENT
	REMAINING DATA SIZE: 237.0 BYTE
SIMULATION TIME: 00:00:16
	PACKET 17 HAS BEEN SENT
	REMAINING DATA SIZE: 222.0 BYTE
SIMULATION TIME: 00:00:17
	PACKET 18 HAS BEEN SENT
	REMAINING DATA SIZE: 207.0 BYTE
SIMULATION TIME: 00:00:18
	PACKET 19 HAS BEEN SENT
	REMAINING DATA SIZE: 192.0 BYTE
SIMULATION TIME: 00:00:19
	PACKET 20 HAS BEEN SENT
	REMAINING DATA SIZE: 177.0 BYTE
SIMULATION TIME: 00:00:20
	COMMAND *CHBTTRY*: Battery level of node x4 is changed to 90
	PACKET 21 HAS BEEN SENT
	REMAINING DATA SIZE: 162.0 BYTE
	NODES & THEIR NEIGHBORS: x1 -> x2, x3 | x2 -> x4, x5 | x3 -> x4 | x4 -> x5, x7 | x5 -> x7 | x6 -> x7, x8 | x7 -> x9 | x8 ->  | x9 ->  |
	5 ROUTE(S) FOUND:
	ROUTE 1: x1 -> x2 -> x4 -> x5 -> x7 -> x9	COST: 4.2020
	ROUTE 2: x1 -> x2 -> x4 -> x7 -> x9	COST: 4.2951
	ROUTE 3: x1 -> x2 -> x5 -> x7 -> x9	COST: 4.1031
	ROUTE 4: x1 -> x3 -> x4 -> x5 -> x7 -> x9	COST: 3.1836
	ROUTE 5: x1 -> x3 -> x4 -> x7 -> x9	COST: 3.2767
	SELECTED ROUTE (ROUTE 4): x1 -> x3 -> x4 -> x5 -> x7 -> x9
SIMULATION TIME: 00:00:21
	PACKET 22 HAS BEEN SENT
	REMAINING DATA SIZE: 147.0 BYTE
SIMULATION TIME: 00:00:22
	COMMAND *RMNODE*: Node x5 is removed
	PACKET 23 HAS BEEN SENT
	REMAINING DATA SIZE: 132.0 BYTE
	NODES & THEIR NEIGHBORS: x1 -> x2, x3 | x2 -> x4 | x3 -> x4 | x4 -> x7 | x6 -> x7, x8 | x7 -> x9 | x8 ->  | x9 ->  |
	2 ROUTE(S) FOUND:
	ROUTE 1: x1 -> x2 -> x4 -> x7 -> x9	COST: 4.2951
	ROUTE 2: x1 -> x3 -> x4 -> x7 -> x9	COST: 3.2767
	SELECTED ROUTE (ROUTE 2): x1 -> x3 -> x4 -> x7 -> x9
SIMULATION TIME: 00:00:23
	PACKET 24 HAS BEEN SENT
	REMAINING DATA SIZE: 117.0 BYTE
SIMULATION TIME: 00:00:24
	COMMAND *INTRUDE*: Node x3 has become a malicious node
	PACKET 25 HAS BEEN SENT
	REMAINING DATA SIZE: 102.0 BYTE
SIMULATION TIME: 00:00:25
	PACKET 26 HAS BEEN SENT
	REMAINING DATA SIZE: 87.0 BYTE
SIMULATION TIME: 00:00:26
	PACKET 27 HAS BEEN SENT
	REMAINING DATA SIZE: 72.0 BYTE
SIMULATION TIME: 00:00:27
	PACKET 28 HAS BEEN DROPPED!
	REMAINING DATA SIZE: 72.0 BYTE
SIMULATION TIME: 00:00:28
	PACKET 28 HAS BEEN SENT
	REMAINING DATA SIZE: 57.0 BYTE
SIMULATION TIME: 00:00:29
	PACKET 29 HAS BEEN SENT
	REMAINING DATA SIZE: 42.0 BYTE
SIMULATION TIME: 00:00:30
	PACKET 30 HAS BEEN SENT
	REMAINING DATA SIZE: 27.0 BYTE
SIMULATION TIME: 00:00:31
	PACKET 31 HAS BEEN DROPPED!
	REMAINING DATA SIZE: 27.0 BYTE
	A POSSIBLE ROUTING ATTACK IS DETECTED! FINDING A NEW ROUTE EXCLUDING THE NODE x3
	NODES & THEIR NEIGHBORS: x1 -> x2, x3 | x2 -> x4 | x3 -> x4 | x4 -> x7 | x6 -> x7, x8 | x7 -> x9 | x8 ->  | x9 ->  |
	1 ROUTE(S) FOUND:
	ROUTE 1: x1 -> x2 -> x4 -> x7 -> x9	COST: 4.2951
	SELECTED ROUTE (ROUTE 1): x1 -> x2 -> x4 -> x7 -> x9
SIMULATION TIME: 00:00:32
	PACKET 31 HAS BEEN SENT
	REMAINING DATA SIZE: 12.0 BYTE
SIMULATION TIME: 00:00:33
	PACKET 32 HAS BEEN SENT
	REMAINING DATA SIZE: 0.0 BYTE
*****************************
AD-HOC NETWORK SIMULATOR - END
*****************************
```


(0,100)
(400,100)
(0,0)
(400,0)
Y
X ->
A
B
C
D
E
F
G
H
I
J